

Chapter 1

Creating Killer iPhone Applications

In This Chapter

- ▶ Figuring out what makes an insanely great iPhone application
 - ▶ Listing the features of the iPhone that can inspire you
 - ▶ Facing the limitations you have to live with
 - ▶ Checking out the possibilities that are open to you
 - ▶ Developing iPhone software now rather than later
-

Imagine that you've just landed at Heathrow Airport. It's early in the morning, and you're dead tired as you clear customs. All you want to do now is find the fastest way to get into London, check into your hotel, and sleep for a few hours.

You take out your iPhone and touch the MobileTravel411 icon. On the left in Figure 1-1, you can see it asks whether you want to use Heathrow Airport as your current location. You touch Yes, and then touch Getting To From (as you can see in the center of Figure 1-1). Because it already knows that you're at Heathrow, it gives you your alternatives. Because of the congestion in and out of London, it suggests using the Heathrow Express, especially during rush hour.

You touch the Heathrow Express tab, and it tells you where to get the train and also tells you that the fare is £14.50 if you buy it from the ticket machine and £17.50 if you buy it onboard the train. (The iPhone on the right in Figure 1-1 is proof that I'm not making this up.) It turns out that you're so jetlagged that you can't do the math in your head, so you touch the Currency button, and it tells you that £ 14.50 is around \$21.35 if you take it from the ATM, \$21.14 on your no-exchange-rate-fee credit card, or \$22.31 at the *bureau de change* at the airport.

Another touch gets you the current weather, which prompts you to dig out a sweater from your luggage before you get on the train.

When you get to Paddington Station, you really don't have a clue where the hotel that someone at the office booked for you might be. You touch Getting Around, and the application allows you to use the hotel address that is in your iPhone Contacts, and then gives you your options when it comes to finally finding that big, comfortable, English bed. These options include walking, taking a taxi, and public transit. You touch Tube, and it directs you to the nearest Tube stop and then displays fares, schedules, and how to buy a ticket.



Figure 1-1:
The Mobile
Travel411
application
can use
your current
location.

How much of a fantasy is this?

Not much. Most of this application already exists. What's more, it took me only a little more than three months to develop that application, starting from where you are now, with no iPhone programming experience.

Creating a Compelling User Experience

What users are after — and what devices like the iPhone enable — is when the separation between them and

- ✓ Other people
- ✓ Tools and information
- ✓ The application itself

is reduced to almost nothing.

These are the three dimensions of what I call “zero degrees of separation.”

We (at least most of us) are social animals, and we feel a need to stay connected with friends, family, and yes, even business associates. What’s truly amazing about the iPhone is that its feature set — its hardware and applications — enables this connectivity seamlessly, ranging from the phone itself to SMS to FaceTime (the iPhone’s new video chat feature) to the built-in camera. Add to that the social networking applications available for your iPhone and your separation from other people reduces even further.

Although one could argue that our present ability to always connect could mean that we have too much of a good thing (especially when it comes to business associates), I’d say that the annoyance factor is a result of the lack of maturity in how we’re using technology. If you think about it, the technology hasn’t really been around all that long, and like most teenagers, it will mature over the next few years.



Many of the applications I’ve just mentioned have long been available on the desktop; what a device like the iPhone does is add mobility to the party. Not only can *you* connect from anywhere, but so can all the people you want to connect to.

This ability to run applications on your phone wherever you are makes it possible to have the information you need (as well as the tools you’d like to use) constantly available. But it’s not just about the fact that the application you need is ready-to-run right there on your phone; it’s (as important) about how the application is designed and implemented. It needs to “work right,” requiring as little as possible from you in terms of effort when it comes to delivering to you what you need.

So, having the app is one thing, but having an optimally designed app is another. These first two dimensions are about what I call *content* — what an application actually does. Another aspect here, of course, is that the potential pool of stuff a user wants to be closer to depends on the *context* in which he’s using the application — where she is and what is going on around a user. In that respect, I like to think of the here-and-nowness — the *relevance* — of the application and information. In plain terms, this means that you want to do a specific task with the help of up-to-date information, which the iPhone can easily access over the Internet through a cell network or Wi-Fi connection. You may even want the information or tasks tailored to where you are, which the iPhone can determine with its location hardware. To use a concrete example, a guidebook application may have a great user interface, for example, but it may not give me the most up-to-date information, or let me know a tour of Parliament is leaving in five minutes from the main entrance. Without those added touches, I’m just not willing to consider an app compelling.

The final dimension is the application itself, or more precisely, how the user interacts with the application. The Multi-Touch user experience sets it up so that the user is naturally more connected to the device — there is no mouse or keyboard acting as an intermediary — and what's more, Apple promotes the use of gestures as much as possible rather than the use of controls. If you want to move a map annotation, don't show arrow or direction controls or force the user type in a new address (although sometimes that may be the way to go) — let the user drag it to where she wants it to be, and provide feedback along the way.

The iPhone allows an immediacy and intimacy as it blends mobility and the power of the desktop to create a new kind of freedom. I like to use the term *user experience* because it implies more than a pretty user interface and nice graphics. A *compelling* user experience enables users to do what they need to do with a minimum of fuss and bother. But more than that, it forces you to think past a clean interface and even beyond basic convenience (such as not having to scroll through menus to do something simple).

Compelling Content in Context — What the App Does

There are a lot of different kinds of applications on the iPhone, ranging from Utility applications like Weather to communication-heavy ones like FaceTime to others that connect you to information and computing power on the Web or even to social networking communities.

These applications shine when they enable you to do what you'd never want to do on the desktop (or even a laptop), either because you don't have the hardware or because, even if you *did* have the hardware, doing it that way would be way too inconvenient. (The added bonus to such applications is that they're available no matter where you are.)

Imagine being at Heathrow, dead tired, taking out your laptop in the middle of a crowded terminal, powering it up, launching the application, and then navigating through it with the touchpad to get the information I got easily while holding the iPhone in one hand. I want that kind of information quickly and conveniently; I don't want to have to dig my way to it through menus or layers of screens (or even going through the hassle of finding a wireless Internet connection). Seconds count. By the time any road warrior tied to a laptop did this at Heathrow, I would already be on the Heathrow Express.

And don't forget games and other pieces of frivolous (that's the point) entertainment to while away the time while waiting in line.

I know what I want, and I want it now, and by the way that's all I want

There are three important things here I want to highlight: relevance, relevance, relevance.

What most of the really good iPhone applications have in common is focus. They address a well-defined task that can be done within a time span that is appropriate for that task. If I need to look something up, I want it right now! If I am playing a game while waiting in line, I want it to be of short duration, or broken up into a series of short and entertaining steps.

The application content itself then — especially for here-and-now applications — must be streamlined and focused on the fundamental pieces of the task. Although you *can* provide a near-infinity of details just to get a single task done, here's a word to the wise: Don't. You need to extract the essence of each task; focus on the details that really make a difference.

Although the travel app I describe at the beginning of this chapter is a good example, a counterexample might also be helpful.

The other night, my wife and I were standing with some friends inside the lobby of a movie theater, trying to decide where to go to grab some dinner. It was cold (at least by California standards), but we wanted to walk to the restaurant from the theater. We had two iPhones going, switching from application to application, trying to get enough information to make a decision. None of the applications gave us what we really needed — restaurants ranked by distance and type, with reviews and directions.

One of the applications was a great example of how to frustrate the user. It allowed you to select a restaurant by distance and cuisine. After you selected the distance, it gave you a list of cuisines. So far, so good. But the cuisine list was not context based; when I tapped *Ethiopian*, all I got was a blank screen. Very annoying! I took it off my iPhone then and there — I don't want an application that makes me work only to receive nothing in return. Your users won't either.

Every piece of a good application is not merely important to the task, but important to *where you are in the task*. For example, if I'm trying to decide how to get to central London from Heathrow, don't give me detailed information about the Tube until I need it.

In some ways, what you leave out is as important to the user experience as what you include.

That doesn't mean that your applications shouldn't make connections that ought to be made. One aspect of a compelling user experience is that all the pieces of an application work together to tell a story. If the tasks in your application are completely unconnected, perhaps they should be separate applications.



An application such as MobileTravel411 is aimed at people who may not know anything about their destination. If the application informs them that one reason to take the Heathrow Express is that it offers convenient Tube access on arrival in London, the users then have a bite-sized piece of valuable information about how to get around London when they're in the city. Save the train routes for when they're in the station.

Finally, limiting the focus to a single task also enables you to leave behind some iPhone (or any other mobile device) constraints such as limited screen size; in this respect, the limitations of the iPhone can guide you to a better application design.

It works the way I do

Great applications are based on the way people — users — think and work. When you make your application a natural extension of the user's world, it makes the application much easier and more pleasant to use — and to master.

Your users already have a mental model that describes the task your software is enabling. The users also have their own mental models of how the device works. At the levels of both content and user interface, your application must be consistent with these models if you want to create a superb user experience (which in turn creates loyalty — to your application).

The user interface in MobileTravel411 was based on how people divide the experience of traveling. Here are typical categories:

- ✓ Foreign currency — how much it really costs and what's the best way to convert money and buy things abroad
- ✓ Getting to and from the airport with maximum efficiency and minimum hassle
- ✓ Getting around a city, especially an unfamiliar one
- ✓ Finding any special events happening while you're in the city
- ✓ Handling traveler's tasks — such as making phone calls, tipping, or finding a bank or ATM — with aplomb
- ✓ Checking the current weather and the forecast
- ✓ Staying safe in unfamiliar territory — places you shouldn't go, what to do if you get in trouble, and so on

This is only a partial list, of course. I get into this aspect of application design in more detail when I take you through the design of MobileTravel411.

I suppose there are other ways to divide the tasks, but anything much different would be ignoring the user's mental model — which would mean that the application would not meet some of the user's expectations. It would be less pleasant to use because it would impose an unfamiliar way of looking at things instead of building on the knowledge and experiences those users already have.



When possible, model your application's objects and actions on objects and actions in the real world. For example, the iPhone has a set of iPod-style playback controls, tapping controls to make things happen, sliding on-off switches, and flicking through the data shown on Picker wheels. All of these are based on physical counterparts in the real world.

Your application's text should be based on the target user. For example, if your user isn't steeped in technical jargon, avoid it in the user interface.

This doesn't mean that you have to “dumb down” the application. Here are some guidelines:

- ✓ If your application is targeted for a set of users who already use (and expect) a certain kind of specialized language, then sure, you can use the jargon in your application. Just do your homework first and make sure that you use those terms *correctly*.

For example, if your application is targeted at high-powered foreign-exchange traders, your application might use *pip* (“price interest point” — the smallest amount that a price can move, as when a stock price advances by one cent). In fact, a foreign-exchange trader expects to see price movement in pips, and not only *can* you, but you *should* use that term in your user interface.

- ✓ If your application requires that the user have a certain amount of specialized knowledge about a task in order to use the application, identify what that knowledge is upfront.
- ✓ If the user is an ordinary person with generalized knowledge, use ordinary language.



Gear your application to your user's knowledge base. In effect, meet your users where they are; don't expect them to come to you.

Disappearing the Technology

Although a few people out there actually love the gadgetry associated with mobile (or any) technology, at the end of the day, when you want to get

something done, you'd really like the technology to disappear. The device itself should become simply an integral part of the environment you're in.

If you base your application on how the user interacts and thinks about the world, designing a great user interface becomes a whole lot easier. But that doesn't mean there aren't a significant number of ways to still blow it.

The user interface — form following function

Don't underestimate the effect of the user interface on the people who are trying to use it. A bad user interface can make even a great application painful to use. If users can't quickly figure out how to use your application, or if the user interface is cluttered or obscure, they're likely to move on and probably complain loudly about the application to anyone who will listen.

Simplicity and ease of use are fundamental principles for all types of software, but in iPhone applications, they are critical. Why? iPhone OS users are probably in the middle of other things while they use your application.

The iPhone hardware and software are outstanding examples of form following function; the user interfaces of great applications follow that principle as well. In fact, even the iPhone's limitations (except for battery life) are a result of form following from the functional requirements of a mobile device user. Just think how the iPhone fulfills the following mobile device user wish list:

- ✓ Small footprint
- ✓ Thin
- ✓ Lightweight
- ✓ Self-contained — no need for an external keyboard or mouse
- ✓ Task-oriented

It's a pretty safe bet that part of the appeal of the iPhone to many people — especially to non-technical users (like most of my friends) — is aesthetic: The device is sleek, compact, and fun to use. But the aesthetics of an iPhone application aren't just about how beautiful your application is onscreen. Eye candy is all well and good, but how well does your user interface match its function — that is, do its job?

Operational consistency

As with the Macintosh, users have a general sense of how applications work on the iPhone. (The Windows OS has always been a bit less user friendly, if you ask a typical Mac user.) One of the early appeals of the Macintosh was how similarly all the applications worked. So Apple (no fools they) carried over this similarity into the iPhone as well. The resulting success story suggests the following word to the wise. . . .



A compelling iPhone user experience usually requires familiar iPhone interface components offering standard functionality, such as searching and navigating hierarchical sets of data. Use the iPhone standard behavior, gestures, and metaphors in standard ways. For example, users tap a button to make a selection and flick or drag to scroll a long list. iPhone users understand these gestures because the built-in applications utilize them *consistently*. Fortunately, staying consistent is easy to do on the iPhone; the frameworks at your disposal have that behavior built in. This is not to say that you should never extend the interface, especially if you're blazing new trails or creating a new game. For example, if you want to “unlock” something in your user interface, why not use a pinch-open gesture, which adheres to the spirit if not the letter of the law, spelling out how to use the “standard” gesture?

Making it obvious

Although simplicity is a definite design principle, great applications are really about being easily understandable to the target user. If I'm designing a travel application, it has to be simple enough for even an inexperienced traveler to use. But if I'm designing an application for foreign-exchange trading, I don't have to make it simple enough for someone with no trading experience to understand.



Keep the following points in mind as you develop an application:

- ✓ The main function of a good application is immediately apparent and accessible to the users it's intended for.
- ✓ The standard interface components also give cues to the users. Users know, for example, to touch buttons and select items from table views (as in the contact application).
- ✓ You can't assume that users are so excited about your application that they're willing to invest lots of time in figuring it out.

Early Macintosh developers were aware of these principles. They knew that users expected that they could rip off the shrink-wrap, put a floppy disk in the machine (these were *really* early Macintosh developers), and do at least something productive immediately. The technology has changed since then; user attitudes, by and large, haven't.

Engaging the user

While I'm on the subject of users, here are two more aspects of a compelling application: direct manipulation and immediate feedback.

- ✓ **Direct manipulation makes people feel more in control.** On the desktop, that meant a keyboard and mouse; on the iPhone, the Multi-Touch interface serves the same purpose. In fact, using fingers gives a user a more immediate sense of control; there's no intermediary (such as a mouse) between the user and the object onscreen. To make this effect happen in your application, keep your onscreen objects visible while the user manipulates them, for example.
- ✓ **Immediate feedback keeps the users engaged.** Great applications respond to every user action with some visible feedback — such as highlighting list items briefly when users tap them.

Because of the limitations imposed by using fingers — especially somewhat pudgy fingers — applications need to be very forgiving. For example, although the iPhone doesn't pester the user to confirm every action, it also won't let the user perform potentially destructive, non-recoverable actions (such as deleting all contacts or restarting a game) without asking, "Are you sure?" Your application should also allow the user to easily stop a task that's taking too long to complete.

Notice how the iPhone uses animation to provide feedback. (I especially like the flipping transitions in the Weather application when I touch the Info button.) But keep it simple; excessive or pointless animation interferes with the application flow, reduces performance, and can really annoy the user.

An app may have an infrastructure the size of Texas behind it, but to the user, it should always appear to be just me and my app.

How the iPhone Makes This All Real

The iPhone's unique software and hardware allow you to create an application that enables the user to do something that may not be practical — or even possible — with a laptop computer. Although the iPhone is a smaller, mobile personal computer, it isn't a replacement for one. It isn't intended to produce documents, proposals, or research. The iPhone has the capability to be an extension of the user, seamlessly integrated into his or her everyday life, and able to accomplish a singly focused task, or step in a series of tasks, in real time, based on where he or she is.

Device-guided design

Although the enormous capabilities of the iPhone make it possible to deliver the compelling user experience your user craves, you must take into account the limitations of the device as well. Keeping the two in balance is *device-guided design*. The next two sections describe both the features and limitations of the iPhone — and how to take them into account as you plan and develop an application. But understanding these constraints can also inspire you to create some really innovative applications. After a closer look at device-guided design, I come back to what makes a compelling user experience.

Exploiting the features

The first thing that comes to mind when you think of an iPhone is mobility. This mobility really has two (separate and distinct) aspects to it:

- ✓ The device is small and unobtrusive enough to have it with you wherever you go. (Pretty obvious, that one.)
- ✓ The device is easily connected. (Look, Ma! No wires.) This fact leads to an interesting corollary: You don't need to have everything stored on the device. All you really need to know how to do is how to jump on the Internet and grab what you need from there.

I'm going to be talking a lot about mobility and what it means for developing iPhone apps during the course of this book, but a number of other hardware and software features built in to the device also enable a rich and compelling experience for the user. Clearly, one of the keys to creating a great application involves taking advantage of precisely these features. In the following sections, I offer a brief survey of the various device features you may want to use. I get into many of them in the book, but let this begin to pique your imagination.

Background processing and posting local notifications

While iOS (the iPhone operating system) doesn't have true multitasking (in fact, unless you have multiple cores or CPUs, no device has it), it does have instant-on task switching that reduces application startup and makes it easier to continue right where you left off. What you also have is the ability for certain kinds of applications to process events in the background. Such applications include

- ✓ **Audio:** The application plays audio in the background.
- ✓ **Location:** The application processes *location events* (information the iOS sends you about changes in location) in the background.
- ✓ **VoIP:** The application provides the ability for the user to make Voice over Internet Protocol calls — turning a standard Internet connection into a way to place phone calls.

What you can also do is post local notifications as a way to get a user's attention when important events happen. (This is especially relevant to Location Services, which I cover in greater detail in Chapter 18.) For example, a GPS navigation-type application running in the background can use local notifications to alert the user when it's time to make a turn. Applications can also schedule the delivery of local notifications for a future date and time and have those notifications delivered even if the application isn't running.

Accessing the Internet

The ability to access Web sites and servers on the Internet allows you to create applications that can provide real-time information to the user. It can tell me, for example, that the next tour at the Tate Modern is at 3 p.m. This kind of access also allows you, as the developer, to go beyond the limited memory and processing power of the device and access large amounts of data stored on servers, or even offload the processing. I don't need all the information for every city in the world stored on my iPhone or have to strain the poor CPU to compute the best way to get someplace on the Tube. I can send the request to a server and have it do all that work.



This is *client-server computing* — a well-established software architecture where the client provides a way to make requests to a server on a network that's just waiting for the opportunity to do something. A Web browser is an example of a client accessing information from other Web sites that act as servers.

Address Book and Contacts

Your application can access the user's contacts on the phone and display that information in a different way, or use it as information in your application. As a user of the MobileTravel411 application, for example, you could enter the name and address of your hotel, and the application would file it in your Contacts database. That way, you have ready access to the hotel address — not only from MobileTravel411, but also from your phone and other applications. Then when you arrive at Paddington Station, the application can retrieve the address from Contacts and display directions for you. What's more, you can also present standard system interfaces for picking and creating contacts in the Address Book.

Calendar Events

If you can leverage the information stored in the Address Book and Contacts databases, it stands to reason you can do the same thing with the Calendar application. You can remind a user when they need to leave for the airport, for example, or create calendar events based on what's happening this week in London. These events show up in the Calendar application and in other applications that support that framework.

Maps and Location

The iPhone OS and hardware allow a developer to determine the device's current location, or even to be notified when that location changes. As people move, it may make sense for your application to tailor itself to where the user is moment by moment.

There are already plenty of iPhone applications that use location information to tell you where the nearest coffeehouse is, or even where your friends are. The MobileTravel411 application uses this information to tell you the nearest Tube stop and give you directions to your hotel.

When you know the user's location, you can even put it on a map, along with other places he or she may be interested in. In Chapter 17, I show you how easy that really is.



An application that provides these kinds of services can also run in the background, and what's more, because using the GPS chip can be a significant drain on the battery, you have access to services that consume less battery power — including a location-monitoring service that tracks significant changes by using only cellular information and the ability to define arbitrary regions and detect boundary crossings into or out of those regions.

Camera and Photo Library

Your application can also access the pictures stored on the user's phone — and by “access” I mean not only display them, but also use or even modify them. The Photos application, for example, lets you add a photo to a contact, and several applications enable you to edit your photos on the iPhone itself. You can also incorporate the standard system interface to actually use the camera as well.

Even more advanced use of the camera is supported. One of the most interesting is *augmented reality*, where you're looking through the iPhone camera at the real world that has computer-generated graphics superimposed. (Actually it's much easier — and cooler — than it sounds.)

Playing audio and video

The iPhone OS makes it easy to play and include audio and video in your application. You can play sound effects or take advantage of the multichannel audio and mixing capabilities available to you. You can also create your own music player that has access to all the audio contents of the user's iPod Library. You can also play back many standard movie file formats, configure the aspect ratio, and specify whether or not controls are displayed. This means that your application can not only use the iPhone as a media player, but also use and control pre-rendered content. And you now can even record and edit HD videos. Keep in mind that if iMovie can do it, you probably can do it, too.

Let the games begin!

Copy, Cut, and Paste operations — and an Edit menu, to boot!

The iPhone OS supports Copy, Cut, and Paste operations within and between applications. It also provides a context-sensitive Edit menu that can display the Copy, Cut, Paste, Select, Select All, and Delete system commands. So that means that while on the iPhone each application is generally expected to play only in its own sandbox, you do actually have ways to send small amounts of data between applications.

Phone, messages (SMS), and mail

It's easy for you to write applications that reduce the degree of separation between your user and their friends (okay, also your business colleagues). Your application can present the standard system interfaces for composing and sending e-mail or SMS messages.

Your app can also interact with phone-based information on devices that have a cellular radio to get information about a user's cellular service provider and be notified when cellular call events occur. You can even access information about active cellular telephone calls, and access cellular service provider information from the user's SIM card. This makes things like VoIP (Voice over Internet Protocol) applications a reality, because you can now do the same things you can do with the cell phone — talk to someone and use the applications on the iPhone at the same time.

User interaction event handling

User interaction events inform your application of user actions — such as when a user touches a view, tilts the device, or presses a button on the headset.

People use their fingers, rather than a mouse, to select and manipulate objects on the iPhone screen. The moves that do the work, called *gestures*, give the user a heightened sense of control and intimacy with the device. There is a set of standard gestures — taps, pinch-close and pinch-open, flicks, and drags — that are used in the applications supplied with the iPhone.



I suggest strongly that you use *only* the standard gestures in your application. Even so, the iPhone's gesture-recognition hardware and software allow you to go beyond standard gestures when appropriate. Because you can monitor the movement of each finger to detect gestures, you can create your own, but use that capability sparingly — only when it's undoubtedly the right thing to do in your application.

But in addition to touches, the user can communicate with the device in another way. The iPhone's new 3-axis gyroscope in combination with the existing accelerometer gives the device what's called *six-axis motion sensing* (x, y, z for both the accelerometer and gyroscope). You can work with both raw accelerometer and gyroscope data, and processed device-motion data (which includes accelerometer, rotation-rate, and altitude measurements). Such data is the basis for many games, as well as the shake-to-undo feature of the iPhone.

Hardware accessories

You can also create new hardware accessories and communicate with them either by a physical connection or Bluetooth. For example, there's a dongle that contains a transmitter you can use to broadcast audio from an iPhone to an FM radio.

Embracing the limitations

The preceding sections make it pretty clear that the iPhone is positively loaded with features you can leverage for use in your own application. All's not *completely* right with the world, however. Along with all those features, the iPhone *does* have some limitations. The key to successful applications — and to not making yourself too crazy — comes in understanding those limitations, living (and programming) within them, and even growing to love them. (It can be done. Honest.) These constraints help you understand the kinds of applications that are right for this device.



Often, it's likely that if you *can't* do something (easily, anyway) because of the iPhone's limitations, then maybe you shouldn't.

So find how to live with and embrace some facts of iPhone life:

- ✓ The small screen
- ✓ Users with fat fingers (me included)
- ✓ Limited computer power, memory, and battery life

The next sections can help get you get closer to this state of enlightenment.

Living with the small screen

While the iPhone's screen size and resolution allow you to deliver some amazing applications, it's still pretty small. Yet while the small screen limits what you can display on a single page, I have managed to do some mental jujutsu on myself to really think of it as a feature.

When your user interface is simple and direct, the user can understand it more easily. With fewer items in a small display, users can find what they want more quickly. A small screen forces you to ruthlessly eliminate clutter and keep your text concise and to the point (the way you like your books, right?).

Designing for fingers

Although the Multi-Touch interface is an iPhone feature, it brings with it limitations as well. First of all, fingers aren't as precise as a mouse pointer, which makes some operations difficult (text selection, for example). User-interface elements need to be large enough (Apple recommends that anything a user

has to select or manipulate with a finger be a minimum of 44 x 44 pixels in size), and spaced far enough apart so that users' fingers can find their way around the interface comfortably.

You also can do only so much by using fingers. There are definitely a lot fewer possibilities when using fingers than the combination of multi-button mouse and keyboard.

Because it's so much easier to make a mistake when using just fingers, you also need to ensure that you implement a robust — yet unobtrusive — undo mechanism. You don't want to have your users confirm every action (it makes using the application tedious), but on the other hand, you don't want your application to let anybody mistakenly delete a page without asking, "Are you *sure* this is what you *really* want to do?" Lost work is worse than tedious.

Another issue around fingers is that the keyboard is not that finger-friendly. I admit it, using the iPhone keyboard is not up there on the list of things I really like about my iPhone. So instead of requiring the user to type some information, Apple suggests that you have a user select an item from a list. But on the other hand, the items in the list must be large enough to be easily selectable, which gets back to the first problem.

But again, like the small screen, this limitation can inspire (okay, may force) you to create a better application. To create a complete list of choices, for example, the application developer is forced to completely understand the context of (and be creative about) what the user is trying to accomplish. Having that depth of understanding then makes it possible to focus the application on the essential, eliminating what is unnecessary or distracting. It also serves to focus the user on the task at hand.

Limited computer power, memory, and battery life

As an application designer for the iPhone, you have several balancing acts to keep in mind:

- ✓ Although significant by the original Macintosh's standards, the computer power and amount of memory on the iPhone are limited.
- ✓ Although access to the Internet can mitigate the power and memory limitations by storing data and (sometimes) offloading processing to a server, those operations eat up the battery faster.
- ✓ Although the power-management system in the iPhone OS conserves power by shutting down any hardware features that aren't currently being used, a developer must manage the trade-off between all those busy features and shorter battery life. Any application that takes advantage of Internet access by using Wi-Fi or the 3G network, core location, and a couple of accelerometers is going to eat up the batteries.

The iPhone OS is particularly unforgiving when it comes to memory usage. If you run out of memory, it will simply shut you down.

This just goes to show that not *all* limitations can be exploited as “features.”

Why Develop iPhone Applications?

Because you can. Because it's time. And because it's fun. Developing my iPhone applications has been the most fun I've had in many years (don't tell my wife!). Here's what makes it so much fun (for me, anyway):

- ✓ **iPhone apps are usually bite-sized — small enough to get your head around.** A single developer — or one with a partner and maybe some graphics support — can do them. You don't need a 20-person project with endless procedures and processes and meetings to create something valuable.
- ✓ **The applications are crisp and clean, focusing on what the user wants to do at a particular time and/or place.** They're simple but not simplistic. This makes application design (and subsequent implementation) much easier — and faster.
- ✓ **The free iPhone Software Development Kit (SDK) makes development as easy as possible.** I reveal its splendors to you throughout this book.

If you can't stand waiting, you *could* go on to Chapter 3, register as an iPhone developer, and download the SDK . . . but (fair warning) jumping the gun leads to extra hassle. It's worth getting a handle on the ins and outs of iPhone application development beforehand.

The iPhone has three other advantages that are important to you as a developer:

- ✓ **The App Store.** Apple will list your application in the App Store, and take care of credit card processing, hosting, downloading, notifying users of updates, and all those things that most developers hate doing. Developers name their own prices for their creations; Apple gets 30 percent of the sales price, with the developer getting the rest.
- ✓ **Apple has an iPhone developer program.** To get your application into the store, you have to pay \$99 to join the program. But that's it. There are none of the infamous “hidden charges” that you often encounter, especially when dealing with credit-card companies. I explain how to join the developer program in Chapter 3 and how to work with the App Store in Chapter 12.
- ✓ **It's a business tool.** The iPhone has become an acceptable business tool, in part because it has tight security, as well as support for Microsoft Exchange and Office. This happy state of affairs expands the possible audience for your application.

Examining the possibilities

Just as the iPhone can extend the reach of the user, the device possibilities and the development environment can extend your reach as a developer. Apple talks often about three different application styles:

- ✓ **Productivity applications use and manipulate information.** The MobileTravel411 application is an example.
- ✓ **Utility applications perform simple, highly defined tasks.** The Weather Application is an example.
- ✓ **Immersive applications are focused on delivering — and having the user interact**

with — content in a visually rich environment. A game is a typical example of an immersive application.

Although these categories help you understand how Apple thinks about iPhone applications (at least publicly), don't let them get in the way of your creativity. You've probably heard ad nauseam about stepping outside the box. But hold on to your lunch; the iPhone box isn't even a box yet. So here's a more extreme metaphor: Try diving into the abyss and coming up with something really new.

The Sample Applications

When I started writing the first edition of this book (which seems like an eternity but was really only 18 months ago), I decided that what I *wasn't* going to do was take the easy way out. By that I mean create a cookbook where you discuss a feature and then show a little bit of code that implements it.

Instead, I wanted to use real-world applications to illustrate the concepts, explain the technology, and give you the kind of background in the static and runtime architecture that you need if you want to develop applications that really exploit the iPhone's feature set.

In Figure 1-2, you can see the first application that I show you how to develop — the one I thought about after I lost my iPhone for the first time. I realized that if anyone found it and wanted to return it, well, returning it wouldn't be easy. Sure, whoever found it could root around in my Contacts or Favorites and maybe call a few of them and ask if any of their friends had lost an iPhone. But to save them the work, I decided to create an application called *ReturnMeTo*, whose icon sat on the upper-left corner of the home screen and looked like something you would want to select if you had found this phone. It would show a phone number to call and create a very happy person at the receiving end of the call.

Originally, I thought I would simply create this application and then get on with the rest of the book. It turned out, however, that as I showed my friends the application, I got a lot of feedback and made some changes to it. I include those changes as well — because they give you insight to the iPhone application-development process. All my friends also told me that they'd love to have the application. (Hey, they're my friends, after all.)

To pay them back for using them as a test group, I uploaded it to the App Store — I show you how to do the same with yours, in detail, in Chapter 12.

Although that app has been eclipsed by some of the newer features rolled out as part of the new iPhone OS 4, it's still a worthwhile exercise. Before, during, or after you've completed it yourself, go to the App Store and download it to your phone to see what this book is about — showing you what you need to do to create your own apps and get them into the App Store.

After I go through developing ReturnMeTo, I take you through the design of MobileTravel411 in Chapter 13. Then I show you how to implement a subset of this application, iPhoneTravel411, which shows you how to use much of the technology that implements the functionality of the MobileTravel411 application. You find out how to use table views (like the ones you see in the Contacts, iPod, Mail, and Settings applications that come with the iPhone), access data on the Web, go out to and return from Web sites while staying in your application, store data in files, include data with your application, allow users to set preferences, and even how to resume your application where the user last left off. I even talk about localization and self-configuring controllers and models. (Don't worry; by the time you get there, you'll know exactly what they mean.)

Because mapping and working with the user's location is such a big part of mobile applications, I show you how easy it is to create custom maps that are tailored to the needs of the user based on what she's doing and where she is. These maps will have annotations (those pins you see in the Maps application that give you information about that location when you tap them), and the annotations will be draggable so the user can change their location and what it says when they are tapped. Finally I show you how to monitor a user's location in the background and notify him or her when they need to know something — even when your app isn't running!

I use real-world applications to show the concepts and give you the background on how things really work on the iPhone — the in-depth knowledge you really need to go beyond the simple "Hello World" apps and create those killer iPhone applications. So be prepared! There may be some places where you might want to say, "Get on with it," but — based on my experience — I'm giving you what you'll need to move from following recipes in a cookbook by rote to modifying and even creating your own recipes.

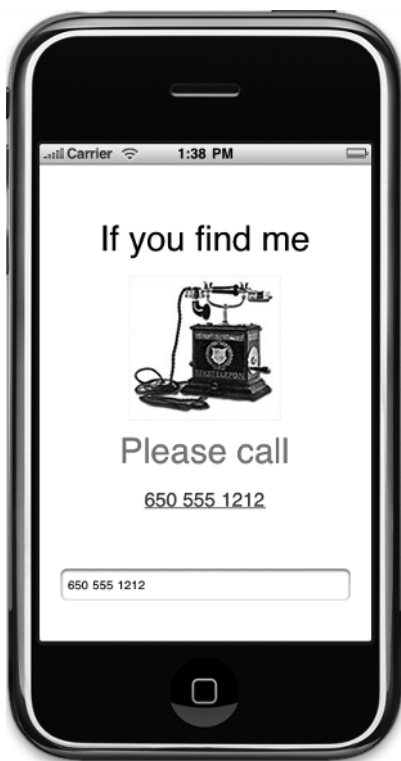


Figure 1-2:
ReturnMeTo
— please!

What's Next

I'm sure that you're raring to go now and just can't wait to download the Software Development Kit (SDK) from the iPhone Developer Web site. That's exactly what I did — and later was sorry that I didn't spend more time upfront understanding how applications work in the iPhone environment.

So I ask you to be patient. In the next chapter, I explain what goes on behind the screen, and then, I promise, it's off to the races.